دانشگاه صنعتی شریف

دانشکده مهندسی برق

گزارش پروژه مدار منطقی

تشخیص اعداد اول

محمد رضا محبوبی اردکانی ۹۲۱۰۲۶۲۱

سعید رفیعیان کوپایی ۹۲۱۰۱۹۹۳

استاد: سرکار خانم دکتر محمدزاده

تاریخ تحویل پروژه: ۲ خرداد ۱۳۹۴

۱ توضیح پروژه

در توضیح پروژه آمده است که، مدار طراحی شده باید یک ۸ بیتی گرفته و در صورتی که این عدد از ۲۰ کوچکتر بود اول بودن یا نبودن آن را معین کند. ما با تغییر اندک، موفق شدیم مداری طراحی کنیم که تمام اعداد اول ۸ بیتی (۰ تا ۲۵۵) را تشخیص دهد.

۲ مقدمات ریاضی

همانطور که می دانیم اعداد اول فقط بر خودشان و ۱ بخش پذیر هستند، و نمی توان آنها را به صورت حاصلضرب اعداد اول کوچکتر از خود نوشت. همچنین طبق ریاضیات می دانیم که اگر عددی اول نباشد، حداقل بر یک عدد اول کوچکتر از جذر خود بخش پذیر است. ما هم از این موضوع استفاده کرده و باقیمانده عدد ورودی را بر ۲، ۳، ۵، ۷، ۱۱، ۱۳ به دست آوردیم. اگر عدد ورودی بر هیچ یک از این ها بخش پذیر نباشد، اول است.

۳ الگوریتم محاسبه باقی مانده

محاسبه باقیمانده تقسیم عدد ورودی بر اعداد گفته شده در بالا از راه تقسیم، روش بسیار پر هزینه ای می باشد، مخصوصا آنکه برای پروگرام کردن این برنامه روی $CPLD$ آزمایشگاه محدودیت ۳۲ ماکروسل وجود دارد. بنابراین از روش زیر برای به دست آوردن باقی مانده تقسیم بدون انجام عمل تقسیم استفاده می کنیم: در این روش با استفاده از یک FSM برای هرعدد که $state$ های آن باقی مانده ها هستند، باقی مانده نهایی را به دست می آوریم. اگر عددی باینری داشته باشیم،اگر بیت w را به سمت راست آن اضافه کنیم، اگر $w = 0$، عدد دو برابر می شود و اگر $w = 1$، علاوه بر دو برابر شدن با ۱ نیز جمع می شود:

$$w = 0 : (x_n x_{n-1} .... x_0 0)_2 = 2 \times (x_n x_{n-1} .... x_0)_2$$

$$w = 1 : (x_n x_{n-1} .... x_0 1)_2 = 2 \times (x_n x_{n-1} .... x_0)_2 + 1$$

با استفاده از همین موضوع ما با شروع از بیت ۷ام تا بیت صفرم، ابتدا بیت ۷ام را به عنوان باقی مانده می گیریم، و در ادامه مطابق جدول زیر (مثال برای ۳) باقی مانده تغییر می کند تا رسیدن به بیت صفرم که باقی مانده نهایی به دست می آید. $state$ ها باقیمانده بر ۳ هستند.

CS	NC	
	$w = 0$	$w = 1$
00	00	01
01	10	00
10	01	10

جدول ۱ : جدول تخصیص حالت برای باقیمانده تقسیم بر ۳

CS	NC	
	$w=0$	$w=1$
000	000	001
001	010	011
010	100	000
011	001	010
100	011	100

جدول ۲: جدول تخصیص حالت برای باقیمانده تقسیم بر ۵

CS	NC	
	$w=0$	$w=1$
000	000	001
001	010	011
010	100	101
011	110	000
100	001	010
101	011	100
110	101	110

جدول ۳: جدول تخصیص حالت برای باقیمانده تقسیم بر ۷

CS	NC	
	$w=0$	$w=1$
0000	0000	0001
0001	0010	0011
0010	0100	0101
0011	0110	0111
0100	1000	1001
0101	1010	0000
0110	0001	0010
0111	0011	0100
1000	0101	0110
1001	0111	1000
1010	1001	1010

جدول ۴: جدول تخصیص حالت برای باقیمانده تقسیم بر ۱۱

CS	NC	
	$w = 0$	$w = 1$
0000	0000	0001
0001	0010	0011
0010	0100	0101
0011	0110	0111
0100	1000	1001
0101	1010	1011
0110	0000	0001
0111	0001	0010
1000	0011	0100
1001	0101	0110
1010	0111	1000
1011	1001	1010
1100	1011	1100

جدول ۵: جدول تخصیص حالت برای باقیمانده تقسیم بر ۱۳

۴ توضیحات کد

برای طراحی (توصیف مدار) از زبان وریلاگ(Verilog) و سطح behavioral استفاده شده است. ورودی های کد عبارتند از: clk: کلاک مدار، $number$: عدد ورودی و $reset$: برای توقف مدار به صورت سنکرون. خروجی های کد عبارتند از: $prime$، not_prime و $gt20$.

$prime = 0$: number is not prime.

$prime = 1$: number is prime.

بلوک $always$ نوشته شده در کد حساس به لبه منفی می باشد. در هر لبه منفی ابتدا $reset$ بودن مدار چک می شود. اگر مدار در حالت $reset$ باشد، مدار متوقف بوده و بیت ۷ام ورودی را به عنوان باقیمانده تقسیم بر ۲، ۳، ۵، ۷، ۱۱، ۱۳ قرار می دهد. اگر مدار در حالت ریست نبود، ورودی را به اندازه مرحله به سمت چپ شیفت داده و بیت ۷ام حاصل را به عنوان ورودی چرخه به دست آوردن باقی مانده قرار می دهیم. اگر عدد ۷ بار شیفت داده شده باشد، از روی باقیمانده بر هر کدام از اعداد فوق اول بودن یا نبودن عدد ورودی مشخص می شود. اگر باقیمانده صفر بود و عدد ورودی خود با اعداد فوق برابر نبود، متغیر های مشخصی ۱ شده و در نهایت اول بودن عدد از nor کردن این متغیر ها معلوم می شود. کد کامپایل شده و طبق گزارش نرم افزار Quartus تعداد ماکروسل های مصرف شده ۲۹ تا می باشد.

در ادامه کد state ها بررسی می شوند، تا مرحله آخر که (بعد از ۷ بار شیفت دادن) باقی مانده نهایی مشخص می شود. کل کد در ادامه قابل مشاهده می باشد.

```
module PrimeNumberDetector(clk, reset, number, prime,
    not_prime, gt20);
input clk, reset;
input [7:0] number;
reg data;
reg [7:0] num;
reg state2;
reg [1:0] state3;
reg [2:0] state5;
reg [2:0] state7;
reg [3:0] state11;
reg [3:0] state13;

reg [3:0] n;
reg r2, r3, r5, r7, r13, r11;

output reg prime, not_prime, gt20;

parameter A = 2'b00, B = 2'b01, C = 2'b10;


always@ (negedge clk) begin
    if (number > 5'h14)
        gt20 = 1'b1;

    if (reset) begin
        state3 <= number[7];
        state5 <= number[7];
        state7 <= number[7];
        state13 <= number[7];
        state11 <= number[7];
        state2 <= number[7];
        n = 4'b0001;
        prime = 1'b0;
        not_prime = 1'b0;
    end
    else begin

        num = number << n;

        if (n == 4'b1000) begin
            r2 = ((state2 == 1'b0) && (number != 8'o2));
            r3 = ((state3 == A) && (number != 8'o3));
            r7 = ((state7 == 3'o0) && (number != 8'o7));
            r13 = ((state13 == 4'h0) && (number != 8'hd));
            r11 = ((state11 == 4'h0) && (number != 8'hb));
            r5 = ((state5 == 3'o0) && (number != 8'o5));

            prime = !(r2 | r3 | r5 | r7 | r11 | r13);
```

```
        not_prime = !prime;
    end

    data = num[7];

    case (state2)
        1'b0:
        begin
            if(data)
                state2 <= 1'b1;
            else
                state2 <= 1'b0;
        end

        1'b1:
        begin
            if(data)
                state2 <= 1'b1;
            else
                state2 <= 1'b0;
        end


        default : state2 <= state2;
    endcase

    case(state3)
        A:
        begin
            if(data)
                state3 <= B;
            else
                state3 <= A;
        end

        B:
        begin
            if(data)
                state3 <= A;
            else
                state3 <= C;
        end

        C:
        begin
            if(data)
                state3 <= C;
            else
                state3 <= B;
        end
```

```
        default: state3 <= state3;
    endcase // 3

    case (state5)
        3'o0:
        begin
            if(data)
                state5 <= 3'o1;
            else
                state5 <= 3'o0;
        end

        3'o1:
        begin
            if(data)
                state5 <= 3'o3;
            else
                state5 <= 3'o2;
        end

        3'o2:
        begin
            if(data)
                state5 <= 3'o0;
            else
                state5 <= 3'o4;
        end

        3'o3:
        begin
            if(data)
                state5 <= 3'o2;
            else
                state5 <= 3'o1;
        end

        3'o4:
        begin
            if(data)
                state5 <= 3'o4;
            else
                state5 <= 3'o3;
        end

        default : state5 <= state5;
    endcase //5


    case(state7)
        3'o0:
```

```
begin
    if(data)
        state7 <= 3'o1;
    else
        state7 <= 3'o0;
end

3'o1:
begin
    if(data)
        state7 <= 3'o3;
    else
        state7 <= 3'o2;
end

3'o2:
begin
    if(data)
        state7 <= 3'o5;
    else
        state7 <= 3'o4;
end

3'o3:
begin
    if(data)
        state7 <= 3'o0;
    else
        state7 <= 3'o6;
end

3'o4:
begin
    if(data)
        state7 <= 3'o2;
    else
        state7 <= 3'o1;
end

3'o5:
begin
    if(data)
        state7 <= 3'o4;
    else
        state7 <= 3'o3;
end

3'o6:
begin
    if(data)
```

```
            state7 <= 3'o6;
        else
            state7 <= 3'o5;
    end

default: state7 <= state7;

endcase // 7


case (state13)
    4'h0:
    begin
        if(data)
            state13 <= 4'h1;
        else
            state13 <= 4'h0;
    end


    4'h1:
    begin
        if(data)
            state13 <= 4'h3;
        else
            state13 <= 4'h2;
    end

    4'h2:
    begin
        if(data)
            state13 <= 4'h5;
        else
            state13 <= 4'h4;
    end

    4'h3:
    begin
        if(data)
            state13 <= 4'h7;
        else
            state13 <= 4'h6;
    end

    4'h4:
    begin
        if(data)
            state13 <= 4'h9;
        else
            state13 <= 4'h8;
```

```
end

4'h5:
begin
    if(data)
        state13 <= 4'hb;
    else
        state13 <= 4'ha;
end

4'h6:
begin
    if(data)
        state13 <= 4'h0;
    else
        state13 <= 4'hc;
end

4'h7:
begin
    if(data)
        state13 <= 4'h2;
    else
        state13 <= 4'h1;
end

4'h8:
begin
    if(data)
        state13 <= 4'h4;
    else
        state13 <= 4'h3;
end

4'h9:
begin
    if(data)
        state13 <= 4'h6;
    else
        state13 <= 4'h5;
end

4'ha:
begin
    if(data)
        state13 <= 4'h8;
    else
        state13 <= 4'h7;
end
```

```
        4'hb:
        begin
            if(data)
                state13 <= 4'ha;
            else
                state13 <= 4'h9;
        end

        4'hc:
        begin
            if(data)
                state13 <= 4'hc;
            else
                state13 <= 4'hb;
        end


        default : state13 = state13;
    endcase // 13


    case (state11)
        4'h0:
        begin
            if(data)
                state11 <= 4'h1;
            else
                state11 <= 4'h0;
        end

        4'h1:
        begin
            if(data)
                state11 <= 4'h3;
            else
                state11 <= 4'h2;
        end

        4'h2:
        begin
            if(data)
                state11 <= 4'h5;
            else
                state11 <= 4'h4;
        end

        4'h3:
        begin
            if(data)
                state11 <= 4'h7;
```

```
    else
        state11 <= 4'h6;
end

4'h4:
begin
    if(data)
        state11 <= 4'h9;
    else
        state11 <= 4'h8;
end

4'h5:
begin
    if(data)
        state11 <= 4'h0;
    else
        state11 <= 4'ha;
end

4'h6:
begin
    if(data)
        state11 <= 4'h2;
    else
        state11 <= 4'h1;
end

4'h7:
begin
    if(data)
        state11 <= 4'h4;
    else
        state11 <= 4'h3;
end

4'h8:
begin
    if(data)
        state11 <= 4'h6;
    else
        state11 <= 4'h5;
end

4'h9:
begin
    if(data)
        state11 <= 4'h8;
    else
        state11 <= 4'h7;
```

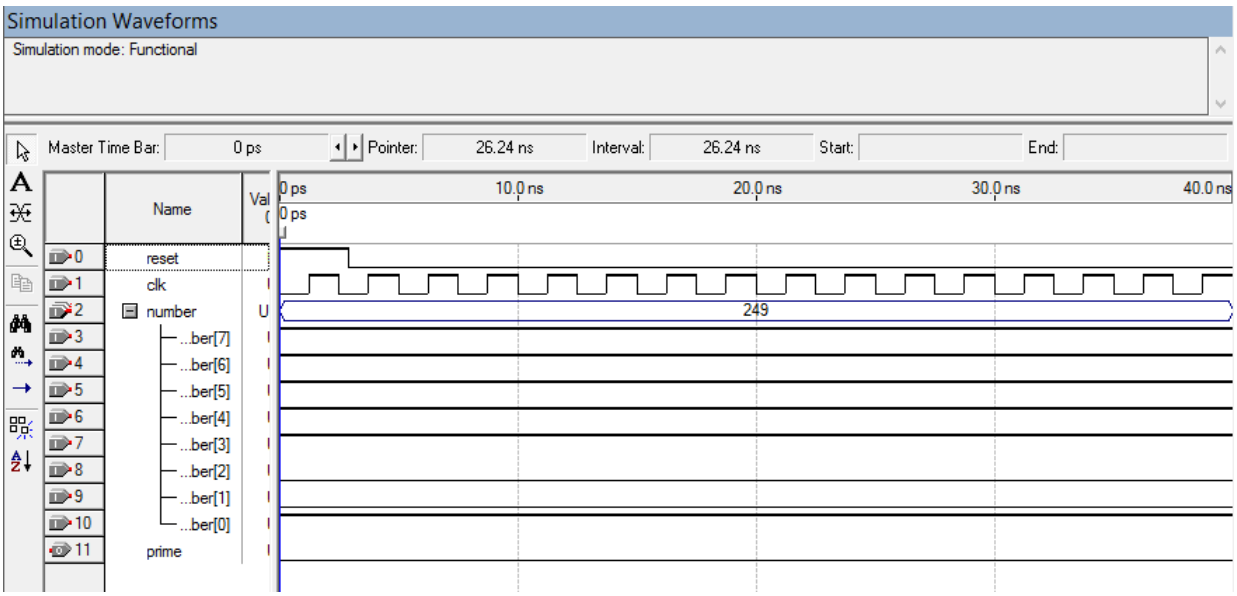
```
            end

            4'ha:
            begin
                if(data)
                    state11 <= 4'ha;
                else
                    state11 <= 4'h9;
            end

            default : state11 = state11;
        endcase // 11




        n = n+1'b1;
    end
end

endmodule
```

۵ نتایج شبیه سازی

شکل ۱: نتیجه شبیه سازی برای عدد ۲۵۱

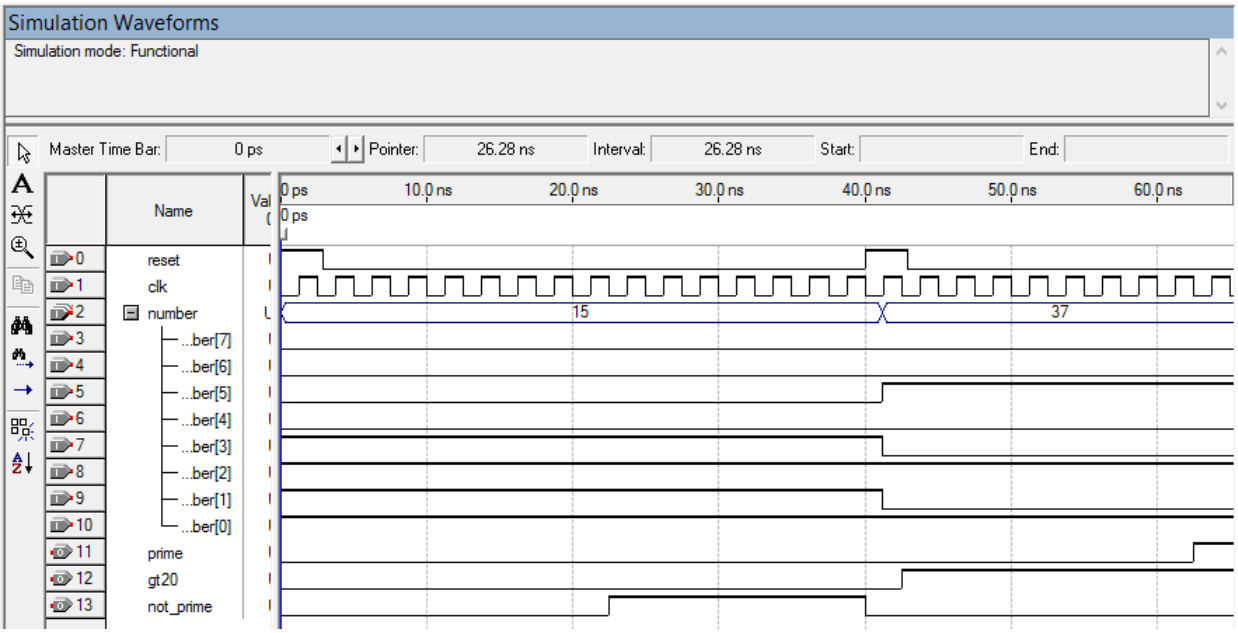
شکل ۲: نتیجه شبیه سازی برای عدد ۱۷۹

همانطور که در نتایج فوق مشخص است بعد از ۸ کلاک از صفر شدن $reset$ سیگنال $prime$، ۱ می شود.



شکل ۳: نتیجه شبیه سازی برای عدد ۲۴۹

همانطور که مشاهده می شود برای اعداد مرکب نیز سیگنال $prime$، هیچگاه ۱ نمی شود.



شکل ۴: نتیجه شبیه سازی برای دو عدد ۱۵ و ۳۷

در نتیجه فوق هر سه خروجی به ازای دو ورودی ۱۵ و ۳۷ آورده شده اند.